UNIVERSIDAD DE MÁLAGA

Modelado y Diseño de Software

Examen Febrero 2022

Nombre: _____

PARTE 1 (15 minutos)

A. Defina los siguientes conceptos [0,-2]
- Ingeniería de software
- Modelo
- Diseño de software
- Patrón de diseño

B. Defina "refactoring" y describa el proceso para implementarlo. [0.25]

C. Defina lo que se entiende por "Clase", "Tipo" y "Plantilla" en Modelado Conceptual. [0.25]

PARTE 2 (3 horas 45 minutos)

P1. Los bancos

Supongamos el sistema financiero de un país, compuesto por bancos.

1) Cada banco tiene sucursales, que pueden estar repartidas por todo el país, pero no puede haber más de tres sucursales de un mismo banco en una ciudad. La estructura de las sucursales de un banco es jerárquica, de manera que una sucursal puede tener a su cargo (subordinadas) algunas otras del mismo banco (ya sean de la misma ciudad o de otras), de cuyas operaciones y clientes también es responsable. [0.25]
2) Los clientes de una sucursal son aquellas personas físicas o empresas que tienen abierta una cuenta en la sucursal. Cada cliente puede tener una o más cuentas en cualquiera de las sucursales de un banco. Cada cuenta sólo puede tener un propietario, que es quien abre la cuenta. Todas las sucursales tienen una cuenta de la que son propietarias, en donde guardan sus activos. [0.25]
3) Cada cuenta tiene un saldo, que debe ser positivo a menos que se trate de una cuenta "de crédito". Este tipo de cuentas permiten tener un saldo que sea negativo, con un límite que se establece al abrirse la cuenta. Un cliente puede solicitar modificar dicho límite a la sucursal donde tiene la cuenta, pero para ello dicha sucursal debe pedir autorización a la sucursal directamente responsable de ella (salvo la central, en la raíz de la jerarquía, que puede tomar decisiones directamente). Las modificaciones del límite de crédito se autorizarán siempre que el nuevo límite sea menor que el anterior, o bien si es solo un 10 % superior al que ya tenía la cuenta y el saldo actual supera el nuevo límite (por ejemplo, si el límite de crédito es 1000 y se pide aumentarlo a 1005 con un saldo de 1100 Euros en la cuenta, la sucursal autorizará dicho cambio de límite). [0.75]
4) Un cliente puede realizar operaciones con su cuenta (pedir el saldo, ingresar o sacar dinero, y transferir dinero a otra cuenta), o bien abrir una cuenta en cualquier sucursal. Al abrir una cuenta, el saldo inicial es 0. Si la cuenta es de crédito, el límite inicial es de 10 Euros. [0.5]
5) Las cuentas tienen una comisión de mantenimiento del 1 % anual. Esto quiere decir que, una vez al año (el 1 de enero), cada sucursal detrae un 1 % del saldo actual de todas sus cuentas. Si su saldo es negativo, transfieren el 1 % del límite de crédito de la cuenta. Ese dinero pasa a ser propiedad de la sucursal, y se almacena en su cuenta. [0.75]
6) Todos los clientes pueden transferir dinero entre cuentas, tanto del mismo banco como de otros. Las transferencias entre cuentas de bancos distintos están grabadas con una comisión del 2 %, es decir, si se transfieren 1000 Euros, en la cuenta destino se ingresan 980. Ese 2 % del dinero pasa a ser propiedad de la sucursales origen y destino de la transferencia a partes iguales (un 1 % para cada una). Las transferencias entre cuentas del mismo banco son gratuitas. [0.75]
7) Las empresas que tengan alguna cuenta con un saldo superior a 1 millón de euros son consideradas VIP por ese banco y cuentan con una serie de ventajas. En primer lugar, en una transferencia entre bancos, la parte correspondiente a la cuenta origen o destino que sea el de ese banco no está grabada con el 1 % correspondiente. En segundo lugar, esas cuentas no pagan comisión de mantenimiento anual. [0.75]
8) Finalmente, las cuentas cuyos propietarios sean sucursales bancarias no pagan ni comisiones anuales ni comisiones de transferencias en ningún banco. [0.5]

Se pide:

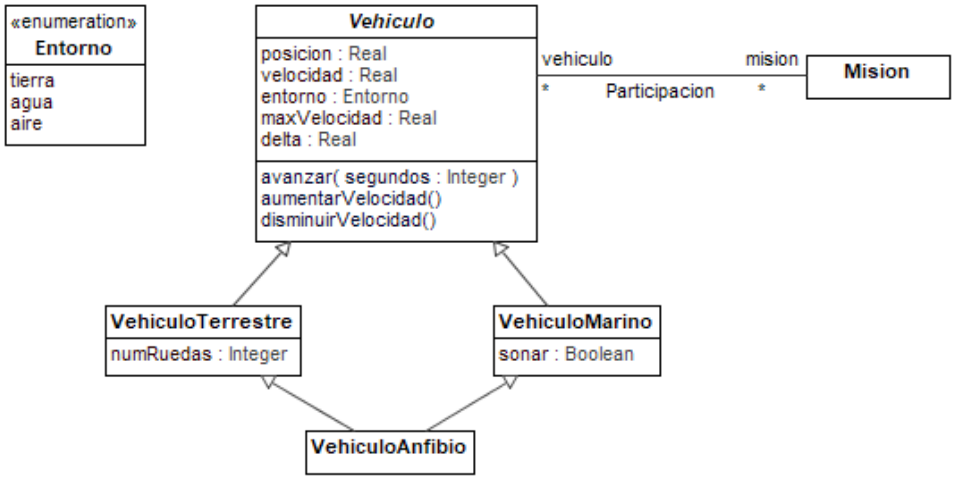
(a) Desarrollar el modelo conceptual de la estructura de dicho sistema utilizando la herramienta USE, incluyendo las restricciones apropiadas que garanticen la coherencia de los datos que maneja el sistema.

(b) Especificar el comportamiento del sistema que permita modelar las operaciones descritas en el enunciado y su funcionamiento. Incluir las pre- y postcondiciones adecuadas en las operaciones especificadas y el cuerpo de las operaciones.

(c) Especificar un sistema con 2 bancos, cada uno con dos sucursales, y cada sucursal con 3 cuentas, una de las cuales ha de ser de crédito. Debe haber al menos dos clientes que tengan varias cuentas en distintos bancos, y una empresa VIP. Hacer transferencias entre las cuentas a lo largo de 3 anualidades, de forma que se cubran todos los casos posibles de cobro y exención de comisiones de mantenimiento y de transferencia, y comprobar que todas las operaciones funcionan de forma correcta. [2]

P2. Vehículos

Supongamos el modelo conceptual mostrado a continuación, que representa diferentes tipos de vehículos. La clase abstracta Vehiculo contiene las características propias de todos ellos, aunque luego cada una las implementa de forma diferente. Los vehículos tienen una posición (por simplicidad supondremos que solo se mueven en línea recta) y pueden moverse a una determinada velocidad (atributo "velocidad", cuyo valor inicial es 0) y en un determinado Entorno (que puede ser, tierra, agua o aire). La operación avanzar() desplaza el vehículo durante el número de segundos indicado en su argumento. Si el vehículo está fuera de su entorno natural no se desplaza, y su velocidad toma el valor 0. Las operaciones aumentarVelocidad y disminuirVelocidad() aumentan o disminuyen la velocidad del vehículo una cantidad determinada ("delta"), que depende del tipo de vehículo de que se trate. Los vehículos pueden participar en misiones.



El modelo permite definir instancias de vehículos terrestres, marinos o anfibios. Los primeros se mueven solo en entornos de tierra, y pueden moverse a una velocidad máxima de hasta maxVelocidad=30 m/s. Incrementan y disminuyen su velocidad en tramos de 5m/s; es decir, cada vez que se invoca su operación aumentarVelocidad() suman delta=5 a su velocidad actual, y cada vez que se invoca su operación disminuirVelocidad() la decrementan en 5. Por su parte, los vehículos marinos solo pueden moverse en entornos de agua, su máxima velocidad es 10 m/s, e incrementan y decrementan su velocidad en tramos de 2 (delta=2). Finalmente, los vehículos anfibios pueden moverse en entornos de tierra o agua, y su comportamiento coincide con el de los vehículos terrestres o marinos dependiendo del entorno en donde se encuentren, cambiando cuando modifica su entorno.

Queremos implementar este modelo conceptual en un lenguaje orientado a objetos como Java, y para ello se pide:

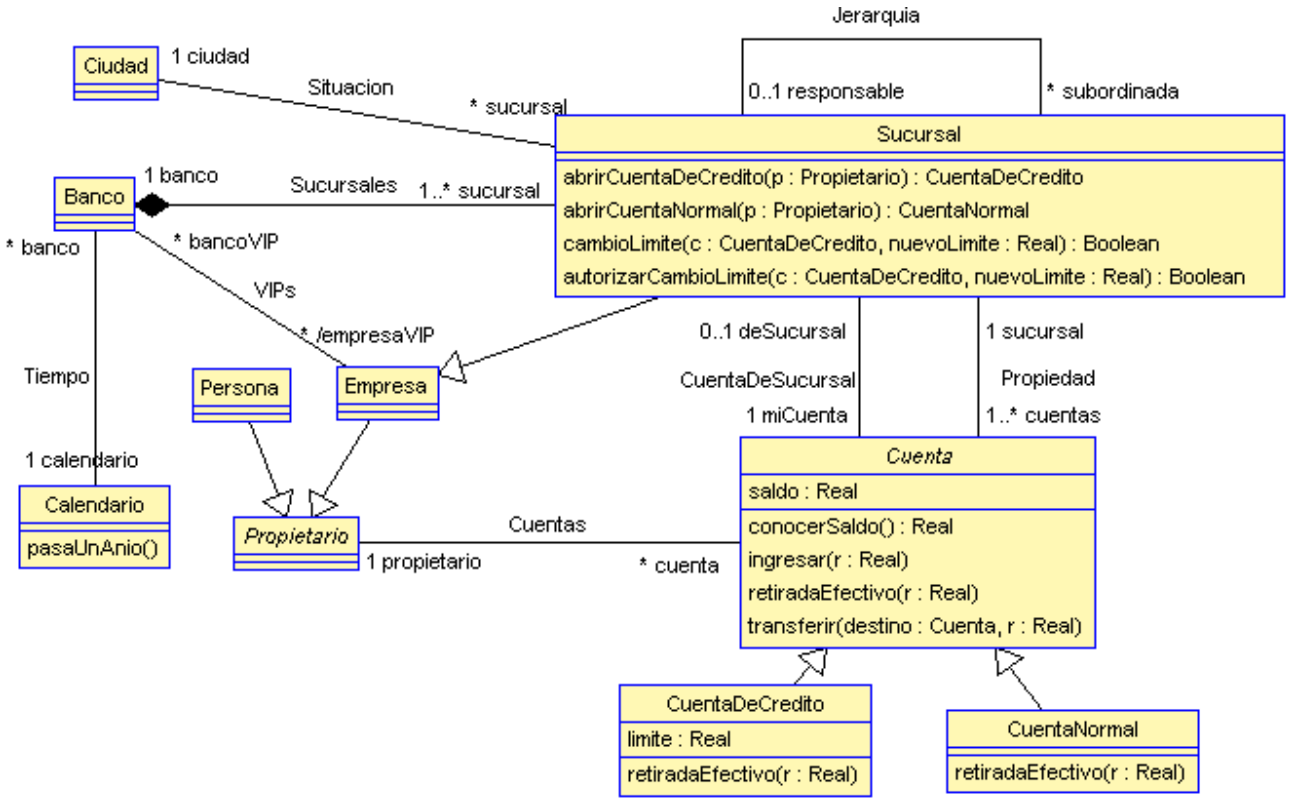
a) Convertir el modelo conceptual indicado arriba en un modelo de diseño, utilizando la herramienta MagicDraw, que permita la implementación, en un lenguaje orientado a objetos como Java, de las entidades, asociaciones y restricciones correspondientes. Incluir en el modelo de diseño explícitamente los correspondientes constructores, getters, setters y demás operaciones que se consideren adecuadas, indicando su visibilidad y argumentos, así como las restricciones OCL oportunas. [1]
b) Desarrollar una implementación en Java correspondiente a dicho modelo de diseño, que implemente las estructuras y funcionalidades tal y como se describen en el modelo de diseño, e incorpore el código de andamiaje para crear objetos y relaciones entre ellos garantizando en todo momento la coherencia de los objetos y de las relaciones de la implementación, y las restricciones e invariantes del modelo, y que no contenga código duplicado. [2]

- **Puntuaciones**: P1 (0.5+4+2) P2 (1+2). Total: 9.5
- **Examen reducido**: Problema P1. Total: 6.5 (nota a multiplicar por 9.5/6.5)
- **Entrega**: La entrega se hará a través del campus virtual, en un solo archivo **en formato PDF**, que contendrá una memoria explicativa del examen y en donde se describirán las soluciones propuestas para cada problema y se incluirán todas las imágenes con los diagramas UML, así como todos los códigos en Java y USE desarrollados.

Soluciones

Problema 1. Apartados (a) y (b)
Desarrollar el modelo conceptual de la estructura de dicho sistema utilizando la herramienta USE, incluyendo las restricciones apropiadas que garanticen la coherencia de los datos que maneja el sistema.

Un modelo que representa ese sistema es el siguiente (con operaciones).



El código USE es el siguiente:

```
model Bancos

class Banco
end

class Sucursal < Empresa
operations
  abrirCuentaDeCredito(p:Propietario):CuentaDeCredito
  begin
      declare c:CuentaDeCredito;
      c:=new CuentaDeCredito();
      insert(self,c) into Propiedad;
      insert(p,c) into Cuentas;
     result:=c;
  end

  abrirCuentaNormal(p:Propietario):CuentaNormal
  begin
      declare c:CuentaNormal;
      c:=new CuentaNormal();
      insert(self,c) into Propiedad;
      insert(p,c) into Cuentas;
      result:=c;
  end
```

```
  cambioLimite(c:CuentaDeCredito,nuevoLimite:Real):Boolean
  begin
      declare autorizado:Boolean;
      result:=false;
      autorizado := if self.responsable->isEmpty() then
                      self.autorizarCambioLimite(c,nuevoLimite)
                    else
                      self.responsable.autorizarCambioLimite(c,nuevoLimite)
                    endif;
      if autorizado then
        c.limite:=nuevoLimite;
        result:=true;
      end
  end
  pre CuentaSubordinada: c.sucursal=self

  autorizarCambioLimite(c:CuentaDeCredito,nuevoLimite:Real):Boolean =
      (nuevoLimite <= c.limite) or
      ((c.limite*1.1 <= nuevoLimite) and (c.saldo >= nuevoLimite))

end -- class Sucursal

composition Sucursales between
  Banco [1]
  Sucursal [1..*]
end

association Jerarquia between
  Sucursal [0..1] role responsable
  Sucursal [*] role subordinada
end

class Ciudad
end

association Situacion between
  Sucursal [*]
  Ciudad [1]
end

abstract class Cuenta
attributes
  saldo:Real init:0.0
operations
  conocerSaldo():Real = self.saldo

  ingresar(r:Real)
    begin
    self.saldo:=self.saldo+r
    end
    pre IngresoPositivo: r>=0
    post IngresoOk: saldo = saldo@pre + r

  retiradaEfectivo(r:Real)
    begin
    self.saldo:=self.saldo-r
    end
    pre IngresoPositivo: r>=0 -- en las subclases se comprueban los limites superiores
    post ExtraccionOk: saldo = saldo@pre - r

  transferir(destino:Cuenta,r:Real)
    begin
    declare sucursalOrigen:Sucursal, sucursalDestino:Sucursal,
          comisionOrigen:Real, comisionDestino:Real;

    comisionOrigen:=r*0.01;         comisionDestino:=r*0.01;
    sucursalOrigen:=self.sucursal;  sucursalDestino:=destino.sucursal;
```

```
      -- exenciones de comisiones
    if sucursalOrigen.banco=sucursalDestino.banco then -- entre cuentas del mismo banco
        comisionOrigen:=0.0; comisionDestino:=0.0;
      else
        if sucursalOrigen.banco.empresaVIP->includes(self.propietario) -- cliente VIP
            or self.deSucursal<>null -- cuenta de sucursal
        then
            comisionOrigen:=0.0;
        end;
        if  sucursalDestino.banco.empresaVIP->includes(destino.propietario) -- cliente VIP
            or destino.deSucursal<>null -- cuenta de sucursal
        then
            comisionDestino:=0.0;
        end;
        sucursalOrigen.miCuenta.ingresar(comisionOrigen);
        sucursalDestino.miCuenta.ingresar(comisionDestino);
      end;

      self.retiradaEfectivo(r);
      destino.ingresar(r-comisionDestino-comisionOrigen);
    end
     pre transferValido: r>=0
    post TransferOk: saldo = saldo@pre - r

end -- class Cuenta

class CuentaNormal < Cuenta
operations
  retiradaEfectivo(r:Real)
    begin
    self.saldo:=self.saldo-r
    end
    pre IngresoPositivo: r>=0 and r <= self.saldo
    post ExtraccionOk: saldo = saldo@pre - r
end

class CuentaDeCredito < Cuenta
attributes
  limite:Real init: 10.0
operations
  retiradaEfectivo(r:Real)
    begin
    self.saldo:=self.saldo-r
    end
    pre IngresoPositivo: r>=0 and r <= self.saldo + self.limite
    post ExtraccionOk: saldo = saldo@pre - r
end

abstract class Propietario
end
class Persona < Propietario
end
class Empresa < Propietario
end

association VIPs between
  Banco [*] role bancoVIP
  Empresa [*] role empresaVIP derive =
Empresa.allInstances()->select(e|e.cuenta->
    exists(c|c.saldo>=1000000 and c.sucursal.banco=self))
end

association Cuentas between
  Propietario [1]
  Cuenta [*]
end
```

```
association CuentaDeSucursal between
  Sucursal [0..1] role deSucursal
  Cuenta [1] role miCuenta
end

association Propiedad between
  Sucursal [1] role sucursal
  Cuenta [1..*] role cuentas
end

class Calendario
operations
  pasaUnAnio()
  begin
    for s in self.banco.sucursal do
        for c in s.cuentas->excluding(s.miCuenta) do
          if not c.sucursal.banco.empresaVIP->includes(c.propietario) then  -- ventaja VIP
              if c.saldo>=0 then
                  c.transferir(s.miCuenta,c.saldo*0.01)
               else
                  c.transferir(s.miCuenta,c.oclAsType(CuentaDeCredito).limite*0.01)
               end
            end
        end
      end
  end
end -- class Calendario

association Tiempo between
  Calendario [1]
  Banco [*]
end

constraints

context Banco inv TresSucursalesDelMismoBanco:
  self.sucursal.ciudad->forAll(c|c.sucursal->select(banco=self)->size()<=3)

context CuentaNormal inv SaldoPositivo:
  self.saldo >= 0

context CuentaDeCredito inv SaldoPositivoCred:
  self.saldo + self.limite >= 0

context Sucursal inv MiCuentaEsMia:
  self.cuentas->includes(miCuenta)

context Sucursal inv JerarquiaOk:
  self.subordinada->closure(subordinada)->excludes(self)

context Sucursal inv JerarquiaOkBancos:
  self.subordinada->forAll(s1,s2|s1.banco=s2.banco)
```

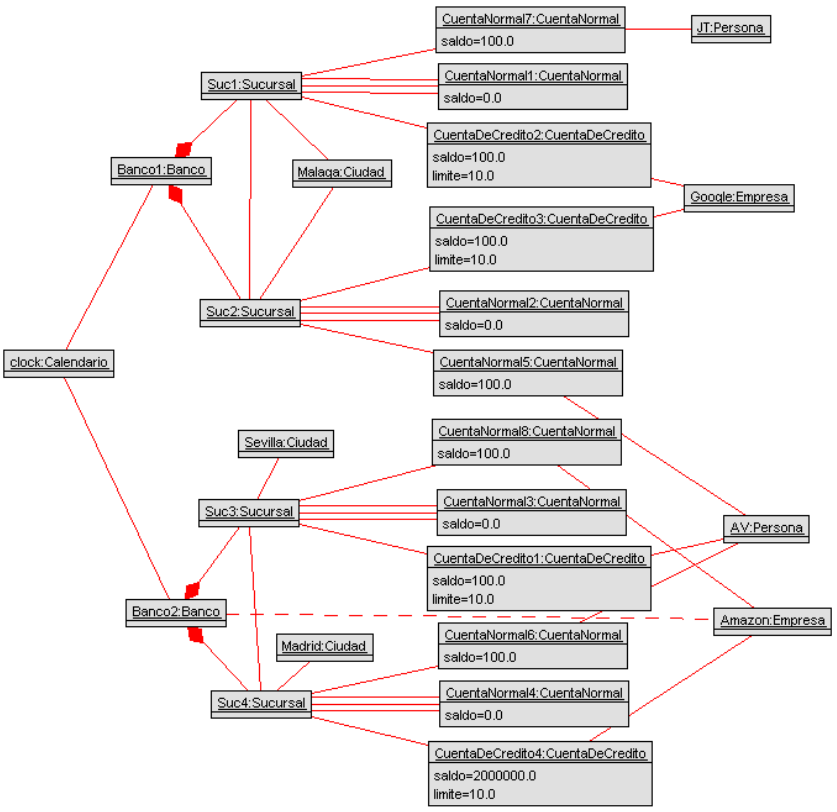
Apartado (b)

La siguiente secuencia de comandos SOIL genera una configuración inicial como la pedida:

```
reset
!new Calendario('clock')
!new Banco('Banco1')
!new Banco('Banco2')
!insert(clock,Banco1) into Tiempo
!insert(clock,Banco2) into Tiempo
!new Ciudad('Malaga')
!new Ciudad('Madrid')
```

```
!new Ciudad('Sevilla')
!new Sucursal('Suc1')
!new Sucursal('Suc2')
!new Sucursal('Suc3')
!new Sucursal('Suc4')
!insert(Suc1,Malaga) into Situacion
!insert(Suc2,Malaga) into Situacion
!insert(Suc3,Sevilla) into Situacion
!insert(Suc4,Madrid) into Situacion
!insert (Banco1,Suc1) into Sucursales
!insert (Banco1,Suc2) into Sucursales
!insert (Banco2,Suc3) into Sucursales
!insert (Banco2,Suc4) into Sucursales
!c:=Suc1.abrirCuentaNormal(Suc1)
!insert(Suc1,c) into CuentaDeSucursal
!c:=Suc2.abrirCuentaNormal(Suc2)
!insert(Suc2,c) into CuentaDeSucursal
!c:=Suc3.abrirCuentaNormal(Suc3)
!insert(Suc3,c) into CuentaDeSucursal
!c:=Suc4.abrirCuentaNormal(Suc4)
!insert(Suc4,c) into CuentaDeSucursal
!new Persona('AV')
!new Persona('JT')
!new Empresa('Google')
!new Empresa('Amazon')
!c:=Suc2.abrirCuentaNormal(AV)
!c:=Suc3.abrirCuentaDeCredito(AV)
!c:=Suc4.abrirCuentaNormal(AV)
!c:=Suc1.abrirCuentaNormal(JT)
!c:=Suc1.abrirCuentaDeCredito(Google)
!c:=Suc2.abrirCuentaDeCredito(Google)
!c:=Suc3.abrirCuentaNormal(Amazon)
!c:=Suc4.abrirCuentaDeCredito(Amazon)
!c.ingresar(1000000)
check
```

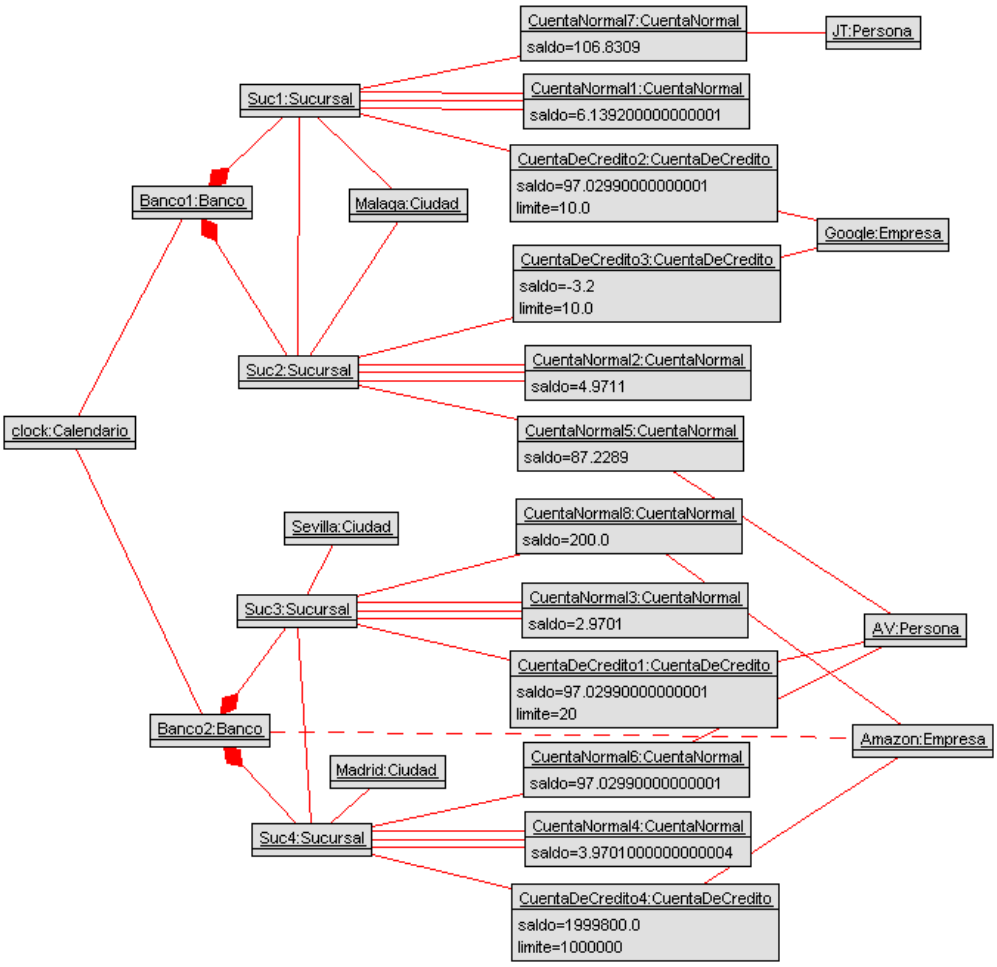
El diagrama de objetos obtenido es el siguiente:



A partir de esa configuración, generamos algunos cambios de limites y transferencias, y pasan 3 años.

```
!r:=Suc4.cambioLimite(c,1000000)
?r
!r:=Suc3.cambioLimite(CuentaDeCredito1,20)
?r
!clock.pasaUnAnio()
!c.transferir(CuentaNormal8,100)
!c.transferir(CuentaNormal6,100)
!CuentaNormal5.transferir(CuentaNormal7,10)
!CuentaNormal6.transferir(CuentaDeCredito3,100)
!CuentaDeCredito3.retiradaEfectivo(200)
!clock.pasaUnAnio()
!clock.pasaUnAnio()
```

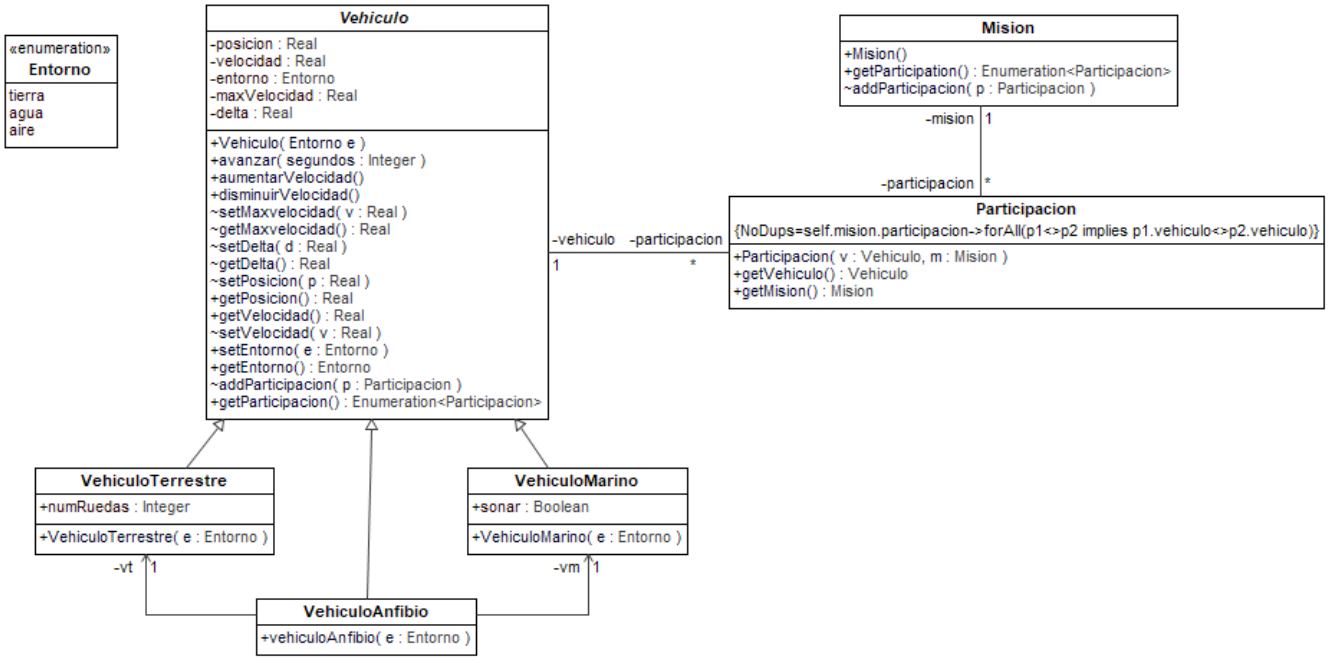
La configuración resultante es la que se muestra a continuación:



Segundo problema

Apartado a)

Para construir el modelo de diseño, en primer lugar es preciso reificar la asociación “Participacion” porque era muchos a muchos. Para modelar la herencia múltiple, lo que hacemos es crear dos asociaciones dirigidas entre VehiculoAnfibio y los otros dos tipos de vehículos, que realmente son sus componentes. La clase VehivuloAnfibio también hereda de Vehiculo, para poder heredar sus metodos. El comportamiento de los métodos de VehiculoAnfibio consiste básicamente en delegar al vehiculo adecuado el correspondiente comportamiento. Dependiendo del entorno en donde se encuentra en un momento u otro, irá invocando a uno u otro.



Apartado b)

```
----------

public enum Entorno { tierra, agua, aire }

----------

//class Mision

import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;

public class Mision {
   private List<Participacion> participacion;

   public Mision() {
      this.participacion = new ArrayList<Participacion>();
   }
   public Enumeration<Participacion> getParticipacion() {
      return java.util.Collections.enumeration(this.participacion);
   }
   void addParticipacion(Participacion p) {
      this.participacion.add(p);
   }
```

```
}

----------

//class Participacion

public class Participacion {

   private Vehiculo vehiculo;
   private Mision mision;

   private boolean noDups(Vehiculo v, Mision m) {
      Enumeration<Participacion> p = v.getParticipacion();
      while(p.hasMoreElements()){
         if (p.nextElement().getMision() == m) return false;
      };
      return true;
   }

   public Participacion(Vehiculo v, Mision m) {
      assert (v!=null);
      assert (m!=null);
      assert this.noDups(v,m):"No se permiten duplicados";
      this.vehiculo = v;
      this.mision = m;
      m.addParticipacion(this);
      v.addParticipacion(this);
   }

   public Vehiculo getVehiculo() {
      return this.vehiculo;
   }

   public Mision getMision() {
      return this.mision;
   }
}

----------

//class Vehiculo

public abstract class Vehiculo {

   private float maxVelocidad;
   private float delta;
   private float posicion;
   private float velocidad;
   private Entorno entorno;
   private List<Participacion> participacion = new ArrayList<Participacion>();


   //Constructor
   public Vehiculo(Entorno e) {
      this.entorno = e;
      this.posicion = 0;
      this.velocidad = 0;
      // maxVelocidad y delta los inicializan las subclases de esta clase Vehiculo (que es abstracta)
   }

   void setMaxVelocidad(float m) {this.maxVelocidad = m;}

   float getMaxVelocidad() {return this.maxVelocidad;}

   void setDelta(float m) {this.delta = m;}

   float getDelta() {return this.delta;}
```

```
   void setPosicion(float p) {this.posicion = p;}

   public float getPosicion() {return this.posicion;}

   void setVelocidad(float v) {this.velocidad = v;}

   public float getVelocidad() {return this.velocidad;}

   public Entorno getEntorno() {return this.entorno;}

   public void setEntorno(Entorno e) {this.entorno=e;} // es publica para poder cambiarlo dinamicamente

   public Enumeration<Participacion> getParticipacion() {
      return java.util.Collections.enumeration(this.participacion);
   }
   void addParticipacion(Participacion p) {
      this.participacion.add(p);
   }

   void moverse(int segundos) { // función auxiliar para actualizar la posicion
      assert(segundos>=0);
      this.setPosicion(this.getPosicion() + this.getVelocidad()*segundos);
   }

   public void avanzar(int segundos) {
      assert(segundos>=0);
      this.moverse(segundos);
   }

   public void aumentarVelocidad() {
      if (this.getVelocidad() < this.getMaxVelocidad())
         this.setVelocidad(Math.min(this.getVelocidad() + this.getDelta(),this.getMaxVelocidad()));
   }

   public void disminuirVelocidad() {
      if (this.getVelocidad() > 0)
         this.setVelocidad(Math.max(this.getVelocidad() - this.getDelta(),0.0f));
   }
}

----------

//class VehiculoMarino

public class VehiculoMarino extends Vehiculo {

   private boolean sonar;

   public VehiculoMarino(Entorno e) {
      super(e);
      this.setMaxVelocidad(10.0f);
      this.setDelta(2.0f);
      this.sonar=false;
   }

   public void setSonar(boolean b) {this.sonar=b;}

   public boolean getSonar() {return this.sonar;}

   public void avanzar(int segundos) {
      assert(segundos>=0);
      if (this.getEntorno()==Entorno.agua) //solo nos podemos mover en agua
         this.moverse(segundos);
   }
}
```

```
----------

//class VehiculoTerrestre

public class VehiculoTerrestre extends Vehiculo {

   private int numRuedas = 0;

   public VehiculoTerrestre(Entorno e, int nRuedas) {
      super(e);
      assert nRuedas >= 0;
      this.setMaxVelocidad(30.0f);
      this.setDelta(5.0f);
      this.numRuedas=nRuedas;
   }

   public void setNumRuedas(int n) {
      assert n>=0;
      this.numRuedas=n;
   }

   public int getNumRuedas() {
      return this.numRuedas;
   }

   public void avanzar(int segundos) {
      assert(segundos>=0);
      if (this.getEntorno()==Entorno.tierra) //solo nos podemos mover en tierra
         this.moverse(segundos);
   }
}

----------

//class VehiculoAnfibio

public class VehiculoAnfibio extends Vehiculo {

   VehiculoTerrestre vt;
   VehiculoMarino vm;
   Vehiculo vehiculoActual;

   public VehiculoAnfibio(Entorno e) {
      super(e);
      vt = new VehiculoTerrestre(Entorno.tierra);
      vm = new VehiculoMarino(Entorno.agua);
      vehiculoActual = (e==Entorno.agua)?vm:vt;
   }

   void setMaxVelocidad(float m) {this.vehiculoActual.setMaxVelocidad(m);}

   float getMaxVelocidad() {return this.vehiculoActual.getMaxVelocidad();}

   void setDelta(float m) {this.vehiculoActual.setDelta(m);}

   float getDelta() {return this.vehiculoActual.getDelta();}

   void setPosicion(float p) {this.vehiculoActual.setPosicion(p);}

   public float getPosicion() {return this.vehiculoActual.getPosicion();}

   void setVelocidad(float v) {this.vehiculoActual.setVelocidad(v);}

   public float getVelocidad() {return this.vehiculoActual.getVelocidad();}

   public void setSonar(boolean b) {this.vm.setSonar(b);}
```

```
    public boolean getSonar() {return this.vm.getSonar();}

    public void setNumRuedas(int n) {this.vt.setNumRuedas(n);}

    public int getNumRuedas() {return this.vt.getNumRuedas(n);

    public void setEntorno(Entorno e) {
       if (this.getEntorno()!=e) { //cambia el entorno, tenemos que cambiar la referencia
          if (e==Entorno.agua) { // cambia a Agua
             vm.setPosicion(this.vehiculoActual.getPosicion());
             vm.setVelocidad(Math.min(this.vehiculoActual.getVelocidad(),vm.getMaxVelocidad()));
             this.vehiculoActual = vm;
          } else {
             if (e==Entorno.tierra) { // cambia a Tierra
                vt.setPosicion(this.vehiculoActual.getPosicion());
                vt.setVelocidad(Math.min(this.vehiculoActual.getVelocidad(),vt.getMaxVelocidad()));
                this.vehiculoActual = vt;
             } else {
                this.setVelocidad(0.0f); //we cannot move in other environments;
             }
          };
          super.setEntorno(e);
       }

       public void avanzar(int segundos) {
          this.vehiculoActual.avanzar(segundos);
       }

       public void aumentarVelocidad() {
          this.vehiculoActual.aumentarVelocidad();
       }

       public void disminuirVelocidad() {
          this.vehiculoActual.disminuirVelocidad();
       }
    }

}
```

ERRORES MÁS COMUNES

Problema 1

- Muchos alumnos asocian las operaciones ingresar(), sacar(), transferir(), etc. a la clase Cliente. En Orientación a Objetos, las operaciones deben asignarse a aquellos objetos a los que modifiquen el estado, y no a quienes normalmente las invocan. Por tanto, esas operaciones deberían asociarse a las Cuentas y no a los Clientes.
- En vez de tener una clase Cuenta y otra CuentaCredito que herede de ella, mejor tener una cuenta abstracta y dos clases que hereden de ella. El problema es que, con la primera de las soluciones cualquier operación que admita una cuenta también admite una cuenta de crédito (por ser un subtipo de ella), y eso hace preciso comprobar los subtipos y hacer muchos castings en las operaciones). Además, las pre- y post-condiciones de la superclase se trasladan a las subclases (un ejemplo es la post-condición de que el saldo de una cuenta sea positivo para poder sacar dinero, lo cual no tiene por qué ser cierto en el caso de la cuenta de crédito).
- Las sucursales no deben heredar de Cliente, sino de Empresa (si esta es una subclase de Cliente).
- Para especificar la cuenta de archivos de una sucursal, muchos han utilizado una asociación entre las clases Cuenta y Sucursal, lo cual es correcto. Sin embargo, la multiplicidad que muchos le han asignado es 1-1, lo cual no es correcto pues esto obliga a que toda cuenta sea una cuenta de activos de alguna sucursal.
- Las cuentas no son VIP. Lo que son VIP son las empresas, y no en general sino para bancos concretos. El hecho de que una empresa sea considerada VIP en un banco Eso se modela mejor con una asociación entre Empresa y Banco, que puede ser derivada (cuando una empresa tenga una cuenta de más de 1MEur en una sucursal, se convierte en VIP del banco al que pertenezca la cuenta).
- En el último apartado, no se usan las operaciones definidas, sino que las cuentas se crean directamente mediante comandos SOIL, se modifican sus saldos alterando directamente el valor de los atributos, etc. El objetivo era comprobar el funcionamiento de las operaciones definidas...

Problema 2

Apartado a)

- En el modelo de diseño, un error común ha sido no reificar la relación Participación, cuando es necesario porque se trata de una asociación muchos a muchos.
- Otro error común ha sido no eliminar la herencia múltiple, sino mantenerla en el diagrama de diseño cuando esto no es implementable en un lenguaje orientado a objetos como Java.

Apartado b)

- El código de andamiaje no asegura la consistencia de las relaciones.
- La idea es usar delegación del vehículo anfibio a sus componentes. Nunca se debe duplicar en el vehículo anfibio el código de cualquiera de los otros vehículos.
- No es buena solución que en el constructor del vehículo anfibio se le pasen por parámetro los vehículos componentes (uno terrestre y otro marino). Es mejor solución que el anfibio cree a sus propios vehículos componentes y los controle él. Si los controla el usuario podría inscribirlos individualmente en misiones (incluso por separado) o gestionarlos mal. Por eso es mejor solución que los cree el vehículo anfibio dentro de su constructor.